ProGreSS: SIMULTANEOUS SEARCHING OF PROTEIN DATABASES BY SEQUENCE AND STRUCTURE[a]

A. BHATTACHARYA T. CAN T. KAHVECI
A. K. SINGH Y.-F. WANG
Department of Computer Science
University of California, Santa Barbara, CA 93106
{*arnab,tcan,tamer,ambuj,yfwang*}*@cs.ucsb.edu*

Abstract

We consider the problem of similarity searches on protein databases based on both sequence and structure information simultaneously. Our program extracts feature vectors from both the sequence and structure components of the proteins. These feature vectors are then combined and indexed using a novel multi-dimensional index structure. For a given query, we employ this index structure to find candidate matches from the database. We develop a new method for computing the statistical significance of these candidates. The candidates with high significance are then aligned to the query protein using the Smith-Waterman technique to find the optimal alignment. The experimental results show that our method can classify up to 97 % of the superfamilies and up to 100 % of the classes correctly according to the SCOP classification. Our method is up to 37 times faster than CTSS, a recent structure search technique, combined with Smith-Waterman technique for sequences.

1 Introduction

The industrialization of molecular biology research has resulted in an explosion of bioinformatics data (DNA and protein sequences, protein structures, gene expression data and genome pathways). Each of these data present a different type of information about the functions of the genes and the interactions between them. Most of the earlier work focuses on only one type of data since each type of data has a different representation and the means of similarity varies for each data type. Combined learning from multiple types of data will help biologists achieve more precise results for several reasons: a) The probability of having false positive results due to errors in data generation decreases since it is less likely for the same error to appear in all the datasets. b) More than one aspect of the biological objects can be captured simultaneously.

1.1 Problem definition

In this paper, we consider the problem of joint similarity searches on protein sequences and structures. A protein is represented as an ordered list of amino acids, where each amino acid has a sequence and a structure component (the terms amino

[a] Work supported partially by NSF under grants EIA-0080134, IIS-9877142, DBI-0213903, and IRI-9908441.

acid and residue are used interchangeably). The sequence component of an amino acid is its residue name indicated by a one letter code from a 20 letter alphabet. The structure component consists of the Secondary Structure Element (SSE) type of that residue (α-helix, β-sheet, or turn), and a 3-D vector which shows the position of its carbon-alpha (C_α) atom.

1.2 Related work

It has been one of the most important goals in molecular biology to elucidate the relationship among sequence, structure and function of proteins[1,2,3]. A handful of algorithms and tools have been developed to analyze sequence and structure similarities between the proteins. These methods are usually focused on either sequence (Smith-Waterman (SW)[4], BLASTP[5,6], PSI-BLAST[7]) or structure information (VAST[8], DALI[9], CE[10], PSI[11], CTSS[12]) for finding similarities between different proteins.

On the other hand, a few tools have been developed for providing integrated environments for analyzing the sequence and structure information together. Protein Analyst[13], 3DinSight[14], and the integrated tools by Huang et al.[15] are among those tools. They provide a combination of separate (but cooperating) programs for integration of sequence and structure analysis under a single working environment. The components of these systems are usually run one after another, with one's results being the input to the other. Although these tools provide integration of multiple types of data, they perform search on only one type of data at a time. We believe that integration of multiple data sources at indexing and search level would provide more precise and efficient tools.

1.3 An overview of our method

We extract a number of feature vectors on sequence and structure components of each protein in the database by sliding a window. Each feature vector maps to a point in a multi-dimensional space. Thus, a protein is represented by a number of points. This multi-dimensional space consists of orthogonal dimensions for sequence and structure. Later, we partition the space with the help of a grid and index these points using Minimum Bounding Rectangles (MBRs).

Given a query, our search method runs in three phases:

Phase 1 (index search): Feature vectors (i.e., points) are extracted from the query protein. For each of these query points, all the database points that are within ϵ_q and ϵ_t distance along the sequence and the structure dimensions are found using the index structure. Each such point casts a vote for the protein to which it belongs as in geometric hashing[16].

Phase 2 (statistical significance): For each database protein, a statistical significance value is computed based on the votes it obtained in Phase 1 and its length.

Phase 3 (post-processing): The top c proteins of highest significance are selected, where c is a predefined cutoff. The optimal pairwise alignment of these c proteins to the query protein are then computed using the SW technique. Finally, the C_α atom of

the matching residues are super-positioned using the least-squares method by Arun et al.[17] to find the optimal RMSD (Root Mean Square Distance).

We name our method *ProGreSS* (*Pro*tein *Gre*p by *S*equence and *S*tructure) since it enables queries based on sequence and structure simultaneously.

The rest of the paper is organized as follows. Section 2 discusses our index structure for proteins. Section 3 explains our search algorithm. Section 4 presents the experimental results. We end with a brief discussion in Section 5.

2 Feature vectors and index construction

In this section, we develop new methods to extract features for protein structures and sequences. Feature vectors for structures are computed as the curvature and torsion values of the residues in a sliding window. Curvature and torsion values provide a necessary and sufficient condition for the isomorphism of two space curves[12]. For a detailed explanation of how curvature and torsion are computed, refer to CTSS[12]. Feature vectors for sequences are computed using a sliding window and a score matrix that defines the similarity between all the amino acids. We also propose a novel index structure to provide efficient access to these features.

2.1 Feature vectors for structure

We slide a window of a prespecified size, w, on the proteins (i.e., each positioning of the window contains w consecutive residues). We will discuss the choice of w later. Figure 1(a) depicts two positionings of the window. For a given window, the curvature and torsion values for each residue in that window is computed. The resulting vector contains $2w$ values since two values are stored per residue in the window. This vector maps to a point in a $2w$-dimensional space. Having a large number of dimensions increases the cost for computing the similarity[18] and the cost for storing the vectors. Therefore, we reduce the number of dimensions to a smaller number, d_t, using the Haar wavelet transformation, at the cost of reduced precision (see[19] for details on Haar transformation). We use d_t = 2 in our experiments. The transformed vector is normalized to $[0,1]^{d_t}$ space. Along with each feature vector, we also store the SSE types of the residues.

As w increases, the feature vector contains information about the correlation between larger number of residues. Thus the similarity between two feature vectors implies longer matches. On the other hand, very large values for w may cause false dismissals since shorter matches may be discarded due to their neighboring residues. We set w = 3 for our experiments.

2.2 Feature vectors for sequence

The similarity between two amino acids of protein sequences is usually defined using score matrices (e.g., PAM and BLOSUM). A score matrix consists of 20 rows and columns; one for each amino acid. The entries of a score matrix denote the score for aligning a pair of residues. If two amino acids are similar, then the score for that pair is large, otherwise it is small.

Given a score matrix M, we call each row of M the *score vector* of the amino acid corresponding to that row. Thus, each entry of this vector shows the similarity of that amino acid to one of the 20 possible amino acids. We define the distance between two amino acids as the Euclidean distance between their score vectors. This is justified, because if the score of aligning two amino acids x and y is high in a score matrix, then they are similar. Therefore, if x is similar (or dissimilar) to another amino acid z, then y is also similar (or dissimilar) to z.

Similar to protein structures, we extract feature vectors for protein sequences by sliding a window of length w (see Figure 1(b) for w = 3). Each positioning of the window contains w amino acids. We append the score vectors of these amino acids in the same order as they appear in the window to obtain a vector of size $20w$. This vector maps to a point in $20w$-dimensional space. Since the number of dimensions is too large for efficient indexing even for small values of w, we reduce the number of dimensions to a smaller number, d_q, using Haar wavelets. Similar to the structure component, we recommend d_q = 2 for optimal quality/time trade-off. The resulting vector is then normalized to $[0,1]^{d_q}$ space. We again choose w = 3.

2.3 Indexing feature vectors

So far we have discussed how to extract feature vectors for structure and sequence components of the proteins separately. In this section, we will discuss how to build an index structure on these feature vectors.

In order to search the protein database based on both sequence and structure, we need to combine the feature vectors for these two components. Since the same window size is used for both the components, every positioning of the window produces one d_t-dimensional feature vector for its structure component and one d_q-dimensional feature vector for its sequence component. We append these two vectors to obtain a single (d_t+d_q)-dimensional vector. The resulting vector is called the *combined feature vector*. Since the entries of each of the feature vectors are normalized to the $[0,1]$ interval, the combined feature vector resides in a $[0,1]^{d_t+d_q}$ *search space*.

The index structure is built by first partitioning the search space into η equal pieces along each dimension. The resulting grid contains $\eta^{d_t+d_q}$ cells of length $1/\eta$ along each dimension. We will discuss the choice of η in Section 3.1. Once the space is partitioned, a window of length w is slid on each protein in the database. For each positioning of the window, the combined feature vector is computed. Each such vector maps to a point p in one of the cells of the grid. For each such point, we check whether that cell is empty. If it is empty, we construct an MBR that contains only p. Otherwise, we find the MBR B in that cell whose volume becomes the smallest after extending it to contain p. If the volume of B, after its expansion, is less than a precomputed volume threshold, V, then we extend B and insert p into B, otherwise we create a new MBR that covers only p. V affects only the performance, not the quality of the search. We chose $V = (1/2\eta)^{d_t+d_q}$ experimentally. Figure 2 presents

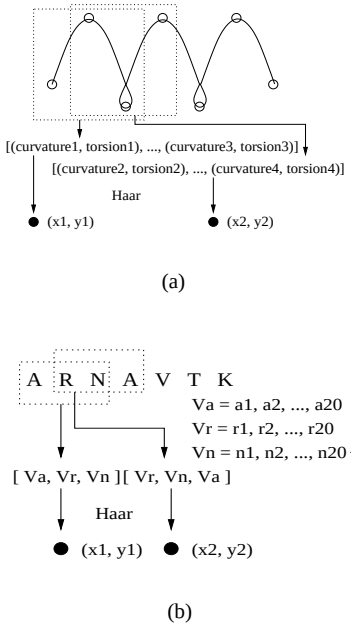

Figure 1: Feature vectors for (a) protein structure, and (b) protein sequence.

```
/* Let D be a dataset that contains proteins,
       w be the window size,
       V be the volume cutoff. */
Procedure CreateIndex(D, w, V)
for each protein x ∈ D
    for each positioning of window of length w
        p := combined frequency vector for current window;
        C := cell that contains p;
        if C = ∅ then
          B.Lower := p;
          B.Higher := p;
          Insert B into C;
        else
          B := argmin_{B∈C}{volume(B ∪ p)};
          if volume(B ∪ p) ≤ V then
            B := B ∪ p;
          else
            B.Lower := p;
            B.Higher := p;
            Insert B into C;
          endif
        endif
    endfor
endfor
```

Figure 2: Algorithm for building the index structure.

the algorithm that constructs the index structure. Figure 3 depicts a layout of a 2-D search space and the MBRs built on the data points for η = 4. Here, $d_t = d_q$ = 1.

3 Query method

Given a query $<Q, \epsilon_q, \epsilon_t, \tau>$, where Q is a query protein, $\epsilon_q \in [0, 1]$ and $\epsilon_t \in [0, 1]$ are the distance thresholds for sequence and structure respectively, and τ is the boolean value regarding the use of SSE information, our search algorithm runs in three phases: 1) index search, 2) statistical significance computation, and 3) post-processing. In this section, we will discuss each of these phases. We will assume that the index structure is built using a user specified score matrix for sequence (e.g., PAM or BLOSUM), and w for the window size.

3.1 Index search

Each residue of the query protein Q consists of a sequence component and a structure component. We extract combined feature vectors from Q by sliding a window of length w on it. Each of these combined feature vectors defines a query point in the search space. Figure 4 shows a sample query point in a 2-D search space, where the horizontal axis is the structure dimension and the vertical axis is the sequence dimension. In this figure, the search space is split into 16 cells numbered from 0 to 15. The query point falls into cell 10. We want to find the database points that are within

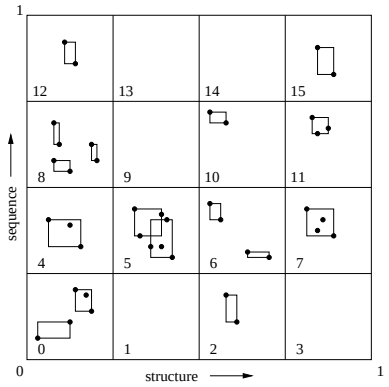

Figure 3: A layout of the MBRs and data points on the search space for η = 4 in 2-D.

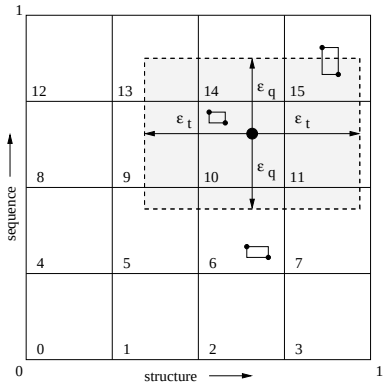

Figure 4: A sample query point and its query box for η = 4 in 2-D.

an ϵ_t distance along the structure dimensions and ϵ_q distance along the sequence dimensions from the query point. In Figure 4, we are interested in the points in the shaded region. Note that if τ = *true*, then we only consider the database points that have the same SSE type as the query point.

For each query point, we construct a query box by extending it by ϵ_t and by ϵ_q in both directions along the structure and the sequence dimensions respectively (see Figure 4). Next, we find the cells in the search space that overlap the query box. If a cell does not overlap the given query box, then it is guaranteed that it does not contain any database points that are in the query box. A cell can overlap a query box in two ways: 1) it is contained in the query box (e.g., cell 10 in Figure 4), or 2) it partially overlaps the query box (e.g., cells 5, 6, 7, 9, 11, 13, 14, and 15 in Figure 4).
1) If a cell is contained in the query box, all the points in that cell are guaranteed to overlap the query box. Therefore, we increment the vote to the database proteins that contains a data point in that cell for each such data point (if τ = *true*, then the vote is added only for the points that have the same SSE type as the query point).
2) If a cell partially overlaps the query box, then we check all the MBRs in that cell. If an MBR is contained in the query box (e.g., the MBR in cell 10), each point in that MBR contributes a vote. If an MBR partially overlaps the query box (e.g., the MBR in cell 15), then we find the points in that MBR that are in the query box to find the votes. If an MBR does not overlap the query box (e.g., the MBR in cell 6), we ignore all the points in that MBR. This method is more precise than geometric hashing [16], because for a given query point it inspects the neighboring cells in addition to the cell into which that query point falls.

The number of partitions η in the search space affects the run time of the index search. As η decreases, each cell contains more MBRs. Therefore, if a query box partially overlaps a cell, then more MBRs need to be tested for intersection with the

query box, thus increasing the search time. On the other hand, having too many partitions have two disadvantages: 1) most of the cells will be sparse or empty incurring space cost. 2) the volume of the boxes will be very small since each cell will get smaller. This increases the total number of MBRs, and hence the number of MBRs for intersection test. From our experiments we recommend η = 10 for optimal results.

3.2 Statistical significance computation

Once the index structure is searched, we obtain a number of votes for each protein in the database. The total number of votes for a protein x shows the number of query points that are close to x's points. We define the *p-value* of a match as the *unexpectedness* of that result. Smaller p-values imply better matches.

Definition 1 *Given a protein x with n points in the index structure and v votes for a given query, the* p-value *of x for that query is defined as the probability of having at least v votes for a randomly generated protein with n points in the search space.*

Next, we discuss the computation of p-values. Consider a protein in the database that is represented in the search space using n points (n = length of protein – window size + 1). Let the protein receive v votes for a given query. Let X be a random variable representing the number of query boxes that overlap with a randomly selected point in the search space. Let μ_X and σ_X^2 be the mean and the variance of X. The total number of query boxes that overlap with n randomly selected points can be computed as $X_n = X + X + \cdots + X$ (exactly n Xs). Since Xs are independent and identically distributed random variables, using Central Limit Theorem, one can show that X_n is normally distributed with mean $\mu_{X_n} = n \cdot \mu_X$ and variance $\sigma_{X_n}^2 = n \cdot \sigma_X^2$. Thus, if μ_X and σ_X^2 are known, one can compute the distribution of X_n using a normal distribution. Since the protein has v votes, its p-value can be computed as $P(X_n \geq v)$.

The computation of p-values requires the values of μ_X and σ_X^2. The distribution of X depends on the distribution of query points, and the distance thresholds ϵ_q and ϵ_t. We compute the values of μ_X and σ_X^2 by generating a large number of random points in the search space and counting the number of query boxes that it overlaps. In our experiments, we generate 10,000 random points for this estimation.

3.3 Post-processing

After the statistical significances of all the proteins are computed, top c proteins with the highest significance are selected as candidates for post-processing, where c is a predefined cutoff. The purpose of post-processing is to find the optimal alignment between the query protein and the most promising proteins. Let q be the query protein. For every protein x in the candidate set, post-processing runs in two steps:

Step 1: We build a $|x| \times |q|$ score matrix, M_{str}, for structure component, where $|x|$ and $|q|$ are the number of residues in x and q, as follows: For each residue in x and q, we construct a 2-D vector as its curvature and torsion. Each entry of M_{str} is then computed as the negative of the Euclidean distance between the <curvature, torsion>-vector of the corresponding residues. For the sequence component, we cre-

ate another $|x| \times |q|$ score matrix, M_{seq}, such that $\forall i, j$ the entry $M_{\text{seq}}[i, j]$ is equal to the score of aligning the i^{th} letter of x with the j^{th} letter of q in the underlying score matrix (e.g., BLOSUM62). The matrices M_{seq} and M_{str} are normalized and a combined score matrix $M_{\text{com}} = (1 - \epsilon_t) \cdot M_{\text{str}} + (1 - \epsilon_q) \cdot M_{\text{seq}}$ is computed. Here, the weights $(1 - \epsilon_t)$ and $(1 - \epsilon_q)$ represent the importance that the user gives to each of the components. The optimal alignment between x and q is then found by running the Smith-Waterman dynamic programming using M_{com}.

Step 2: The alignment obtained in Step 1 defines a one-to-one mapping between a subset of residues of x and q, and is optimal with respect to M_{com}. Finally, we find the 3-D rotation and translation of x that gives the minimum RMSD to q by using a least-squares fitting method[17].

4 Experimental evaluation

We used single domain chains in our experiments. We downloaded all the protein chains in PDB (`http://www.rcsb.org/pdb`) that contain only one domain according to VAST[8] and SCOP[20] classifications. We only considered proteins that are members of one of the following SCOP classes: all α, all β, α+β and α/β. We identified the superfamilies (according to SCOP classification) that have at least 10 representatives in this dataset. There are 181 such superfamilies. We created a database D of size 1810 proteins by including 10 proteins from each of these superfamilies. We formed a query set, D_Q, by choosing a random chain from each of the 181 superfamilies in D. D_Q is large enough to sample D since it contains one protein from each superfamily. We ran a number of experiments on these sets to test the quality and the performance of ProGreSS. The tests were run on a computer with two AMD Athlon MP 1600+ processors with 2 GB of RAM, running Linux 2.4.19.

In the rest of this section, we use w for the window size, c for the cutoff, ϵ_t and ϵ_q for the structure and sequence distance thresholds, τ for the SSE type match choice, and η for the number of partitions. We employ the BLOSUM62 score matrix for sequences in all of our experiments. The number of dimensions d_q and d_t for sequence and structure are both set to 2.

4.1 Quality test

Our first experiment set inspects the effect of various indexing and search parameters on the quality of our index search results. We classify a given query protein into one of the superfamilies and classes using the c best seeds as follows. The logarithms of the p-values of the matches in top c results in each superfamily are accumulated. The query protein is classified into the superfamily that has the largest magnitude of this sum. We use the same technique to classify the query protein to one of the four SCOP classes: all α, all β, α+β and α/β. Since the queries are selected from the database, in order to be fair, we do not take into account the query protein itself if it is among the top c results. We will only report the results for τ = *true*, since it usually produced slightly better results than τ = *false*.

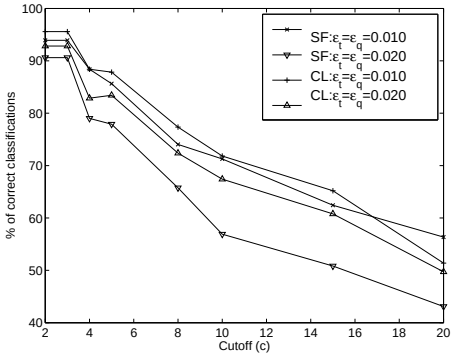

Figure 5: Percentage of query proteins correctly classified for different values of c.

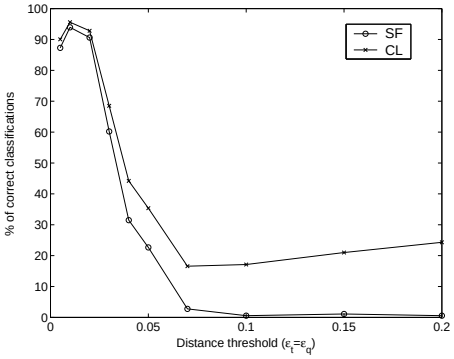

Figure 6: Percentage of query proteins correctly classified for different values of distance threshold when $\epsilon_t = \epsilon_q$.

Figure 5 shows the percentage of query proteins correctly classified to classes (CL) and superfamilies (SF) for different values of c, where $\epsilon_t = \epsilon_q = 0.01$ and 0.02, and $w = 3$. In all these experiments, we obtained the best results for $c = 2$ and 3. We achieved up to 96 % and 94 % correct classification for classes and superfamilies respectively. As c increases, our method starts retrieving proteins from other classes and superfamilies. We set $c = 3$ for the rest of the experiments.

Figure 6 plots the percentage of correctly classified proteins for varying distance thresholds when $\epsilon_t = \epsilon_q$ and $w = 3$. The purpose of this experiment is to understand what a good distance threshold should be when sequence and structure have equal importance. The graph shows that the accuracy of ProGreSS increases when distance threshold increases from 0.005 to 0.01. At $\epsilon_t = \epsilon_q = 0.01$, ProGreSS achieves 96 % and 94 % correct classification for classes and superfamilies. As the distance threshold increases, ProGreSS starts retrieving distant proteins and its accuracy drops.

Figure 7 shows the percentage of correctly classified superfamilies for different values of ϵ_t when ϵ_q is fixed and for different values of ϵ_q when ϵ_t is fixed, for $w = 3$. This experiment shows the effect of distance threshold for each of the structure and sequence components separately. When ϵ_q is fixed, as ϵ_t decreases, the classification quality of ProGreSS increases. This implies that our method can find better results when the distance threshold is small. The highest accuracy obtained is 62 %. For $\epsilon_q = 1.0$ (i.e., when the sequence component is ignored), ProGreSS performs the worst. This is an important result since it shows that searches based on structure alone would incur more false positives than the searches based on both sequence and structure. When ϵ_t is fixed, as ϵ_q decreases, ProGreSS classifies more proteins correctly. In this case, 94 % of the proteins are correctly classified into their superfamilies. Our method performs the worst when $\epsilon_t = 1.0$. This result leads to two important conclusions: 1) Searching by sequence information alone is worse than searching based by

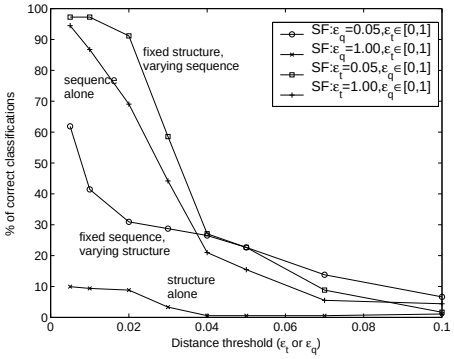

Figure 7: Percentage of query proteins correctly classified for different values of ϵ_t (ϵ_q) when ϵ_q (ϵ_t) is fixed.

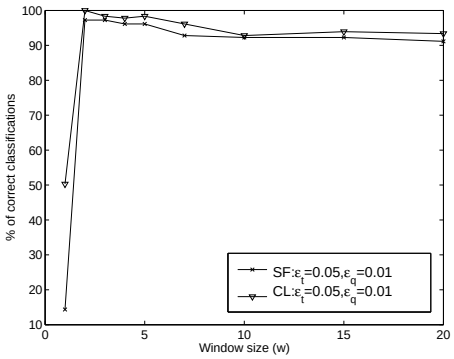

Figure 8: Percentage of query proteins correctly classified for different values of w.

sequence and structure simultaneously. 2) For purposes of classification, our extraction of feature vectors for sequence is better than those for structure.

Figure 8 plots the effect of window size on the classification quality of ProGreSS. The best results are achieved at w = 3. At this window size, ProGreSS can classify 100 % and 97 % of the classes and superfamilies correctly. ProGreSS performs worse for smaller window sizes since correlations between the consecutive residues are not reflected to the index structure. As w becomes larger than 3, ProGreSS starts to miss some of the good results since shorter local matches are not preserved for large w.

Finally, Figure 9 compares the accuracy of our technique with CTSS, a recent algorithm that considers structure alone. We show the number of correct proteins (those from the same superfamily as the query protein) for different values of c. CTSS finds 3 out of 10 correct proteins in the first 100 candidates. On the other hand, our method finds the same number of proteins within the first 4 candidates.

4.2 Performance test

In this experiment set we compare the performance of our method to CTSS. In order to have fair results, we run CTSS in two phases: 1) the top c candidates are found using the original CTSS code and each candidate is aligned to the query by using SW based on its structure score matrix. 2) The optimal sequence alignment of all the database proteins to the query are determined using SW alignment. For CTSS and ProGreSS, we choose c = 100 and 4 respectively. This is because the quality of their candidates are similar for these values of c (see Figure 9). We run queries for all of the 181 proteins and align only the candidate proteins to each of the query proteins.

Figure 10 shows the average time spent by CTSS and our method. The run times for CTSS and SW are 38 and 18 seconds respectively. The graph for CTSS+SW is flat since these methods are independent of η. ProGreSS runs faster than CTSS+SW for all values of η. For η = 10, ProGreSS runs in only 1.5 seconds (i.e., 37 times faster

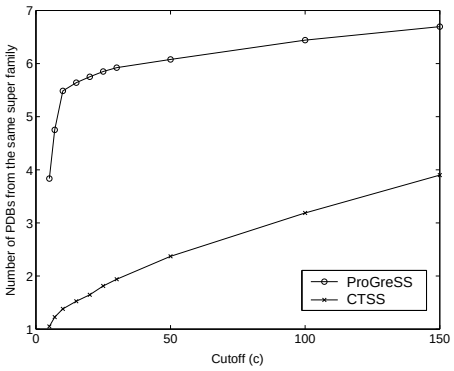

Figure 9: Number of proteins found from the same superfamily as the query protein for ProGreSS and CTSS for different values of c.

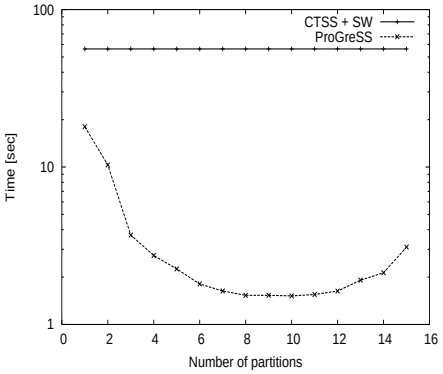

Figure 10: Comparison of running times of ProGreSS and CTSS+SW.

than CTSS+SW). As η gets smaller, ProGreSS runs slower. This is because when a query box partially overlaps a cell, more MBRs are tested for intersection. As η becomes larger than 10, the performance of ProGreSS drops since the total number of MBRs in the index structure increases.

5 Discussion

In this paper, we considered the problem of joint similarity searches on protein sequences and structures. We proposed a sliding-window–based method to extract feature vectors on the sequence and structure components of the proteins. Each feature vector is mapped to a point in a multi-dimensional space. We developed a novel index structure by partitioning the space with the help of a grid, and clustering these points using Minimum Bounding Rectangles (MBRs). Our search method finds the number of feature vectors that are similar to the feature vectors of a given query for each database protein. We also proposed a new statistical method to compute the significance of the results found at the index search phase. The results are sorted according to their significance and the most promising results are aligned using the Smith-Waterman (SW) method[4] and the least-squares method by Arun et al.[17] to find the optimal alignment.

According to the experimental results on a set of representative query proteins, ProGreSS classified all of the classes and 97 % of the superfamilies correctly. Our method ran 37 times faster than CTSS, a recent structure search technique, combined with the SW technique for sequences.

Combined learning from multiple data sources is an important research problem since each data provides a correlated yet different type of information about the protein. ProGreSS provides the user a wide flexibility of search parameters to assign weights on each of these data types. We believe that, the methods discussed in this

paper are an important step in understanding the functions of proteins better, and will be widely applicable in the area of proteomics. In the future, we would like to include other features into our index structure such as expression arrays and pathways.

References

1. T. C. Wood and W. R. Pearson. Evolution of Protein Sequences and Structures. *J. of Mol. Biol*, 291(4):977–995, 1999.
2. H. Hegyi and M. Gerstein. The Relationship between Protein Structure and Function: a Comprehensive Survey with Application to the Yeast Genome. *J. of Mol. Biol.*, 288(1):147–164, 1999.
3. J. M. Sauder, J. W. Arthur, and R. L. Dunbrack Jr. Large-scale comparison of protein sequence alignment algorithms with structure alignments. *Proteins: Structure, Function, and Genetics*, 40(1):6–22, 2000.
4. T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *J. of Molecular Biology*, March 1981.
5. S. Altschul, W. Gish, W. Miller, E. W. Meyers, and D. J. Lipman. Basic local alignment search tool. *J. Molecular Biology*, 215(3):403–410, 1990.
6. W. Gish and D.J. States. Identification of protein coding regions by database similarity search. *Nature Genet.*, pages 266–272, 1993.
7. S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucl. Acids. Res.*, 25(17):3389–3402, 1997.
8. T. Madej, J.-F. Gibrat, and S.H. Bryant. Threading a database of protein cores. *Proteins*, 23:356–369, 1995.
9. L. Holm and C. Sander. Protein structure comparison by alignment of distance matrices. *Journal of Molecular Biology*, 233:123–138, 1993.
10. H.N. Shindyalov and P.E. Bourne. Protein structure alignment by incremental combinatorial extension (CE) of the optimal path. *Protein Engineering*, 11(9):739–747, 1998.
11. O. Çamoğlu, T. Kahveci, and A. K. Singh. Towards Index-based Similarity Search for Protein Structure Databases. In *CSB*, 2003.
12. T. Can and Y. F. Wang. CTSS: A Robust and Efficient Method for Protein Structure Alignment Based on Local Geometrical and Biological Features. In *CSB*, 2003.
13. M. A. S. Saqi, D. L. Wild, and M. J. Hartshorn. Protein Analyst - a distributed object environment for protein sequence and structure analysis. *Bioinformatics*, 15:521–522, 1999.
14. J. An, T. Nakama, Y. Kubota, H. Wako, and A. Sarai. Construction of an Integrated Environment for Sequence, Structure, Property and Function Analysis of Proteins. *Genome Informatics*, 10:229–230, 1999.
15. C. C. Huang, W. R. Novak, P. C. Babbitt, A. I. Jewett, T. E. Ferrin, and T. E. Klein. Integrated Tools for Structural and Sequence Alignment and Analysis. In *PSB*, pages 227–238, 2000.
16. H.J. Wolfson and I. Rigoutsos. Geometric hashing: An introduction. *IEEE Computational Science & Engineering*, pages 10–21, Oct-Dec 1997.
17. K.S. Arun, T.S. Huang, and S.D. Blostein. Least-squares fitting of two 3-D point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(5):698–700, September 1987.
18. K. S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When Is "Nearest Neighbor" Meaningful? In *ICDT*, pages 217–235, 1999.
19. R.M. Rao and A.S. Bopardikar. *Wavelet Transforms Introduction to Theory and Applications*. Addison Wesley, 1998.
20. A. G. Murzin, S. E. Brenner, T. Hubbard, and C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J. Mol. Biol.*, 247:536–540, 1995.